МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЕТ

по лабораторной работе №2
по дисциплине «Алгоритмы и структуры данных »
Тема: Иерархические списки

Студент гр. 8304 _____ Бутко А.М.

Преподаватель _____ Фирсов М.А.

Санкт-Петербург
2019

Цель работы.

Изучить динамические структуры данных, иерархический список, бинарное коромысло. Попробовать реализовать специальный иерархический список.

Задание.

4 вариант.

Говорят, что бинарное коромысло сбалансировано, если момент вращения, действующий на его левое плечо, равен моменту вращения, действующему на правое плечо (то есть длина левого стержня, умноженная на вес груза, свисающего с него, равна соответствующему произведению для правой стороны), и если все подкоромысла, свисающие с его плеч, также сбалансированы. Написать рекурсивную функцию или процедуру Balanced, которая проверяет заданное коромысло на сбалансированность (выдает значение true, если коромысло сбалансировано, и false в противном случае).

Основные теоретические положения.

Бинарное коромысло устроено так, что у него есть два плеча: левое и правое. Каждое плечо представляет собой (невесомый) стержень определенной длины, с которого свисает либо гирька, либо еще одно бинарное коромысло, устроенное таким же образом.

В соответствии с данным выше рекурсивным определением бинарного коромысла представим бинарное коромысло списком из двух элементов

БинКор ::= (Плечо Плечо),

где первое плечо является левым, а второе – правым. В свою очередь Плечо будет представляться списком из двух элементов

Плечо ::= (Длина Груз),

где Длина есть натуральное число, а Груз представляется вариантами

Груз ::= Гирька | БинКор,

где в свою очередь Гирька есть натуральное число. Таким образом, БинКор есть специального вида иерархический список из натуральных чисел.

Описание алгоритма.

Была реализована структура BinaryScales, в которой будет храниться информация о бинарном коромысле. В структуру вложен конструктор, который инициализирует данные “по умолчанию”, а так же внутри хранятся поля структуры: atomNumber (собственный номер элемента (атома)), atomLeftNumber и atomRightNumber (номера элементов по левое и по правое плечо коромысла соответственно), atomArmLength (длина плеча коромысла), atomKey (значение элемента – 0, если элемент – бинарное коромысло, натуральное число, если элемент – груз), isChecked и isBalanced(информация о проходе функции balance и балансирует ли подкоромысло в балансе соответственно), указатели prevAtom, leftAtom, rightAtom (на предыдущий, левый и правый элемент соответственно). (Здесь и далее будем называть иерархический список “деревом”, т. к. так легче визуализировать себе бинарное коромысло).

Функцией addAtom происходит добавление элемента в будущее бинарное коромысло. Для корректности введенных данных и последующей работе программы нумерацию элементов нужно производить по левой стороне, затем подниматься по подкоромыслу и производить нумерацию правых элементов, не менять порядок элементов при вводе.

После того как все элементы будут добавлены, необходимо с помощью функции backToZero вернуть указать на нулевой элемент для последующего обхода созданного специального иерархического списка.

Функцией balance производится спуск по дереву до “конечных” грузов (сначала по левой, затем аналогично по правой стороне бинарного коромысла), высчитывается момент вращения правого и левого грузов. На основе сравнения заполняем поле структуры isChecked, поднимаемся указателем на уровень выше, вписываем в atomKey подкоромысла сумму масс грузов (обобщаем коромысло до понятия “груз”). Ответ на задачу выводим в консоль и в файл.

Описание основных данных.

Рассмотрим ввод и получившиеся иерархическое дерево на примере одного из тестов:

0 1 4

1 2 3 0 3

2 0 0 2 1

3 0 0 1 2

4 0 0 1 9

Иерархия элементов изображена на рисунке 1, получившееся бинарное коромысло изображено на рисунке 2, принцип адресации изображена на рисунке 3.

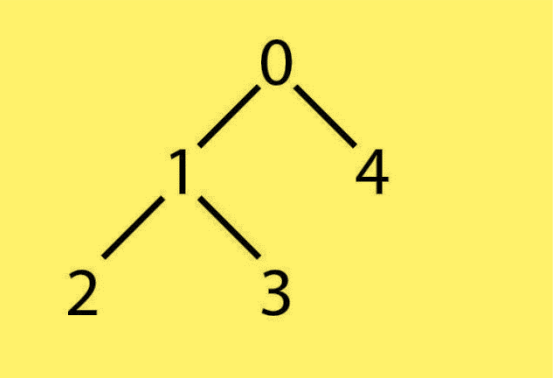

рис.1

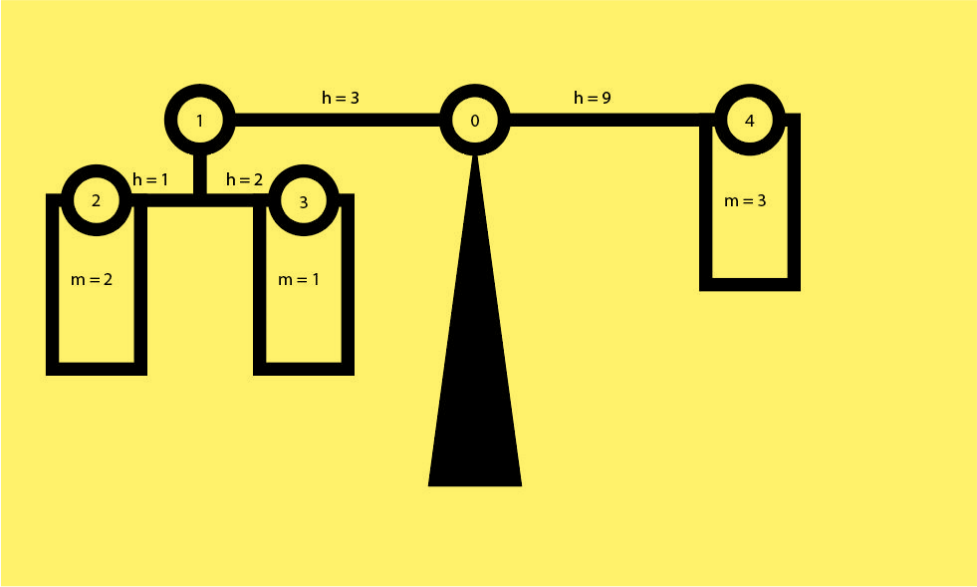

рис. 2

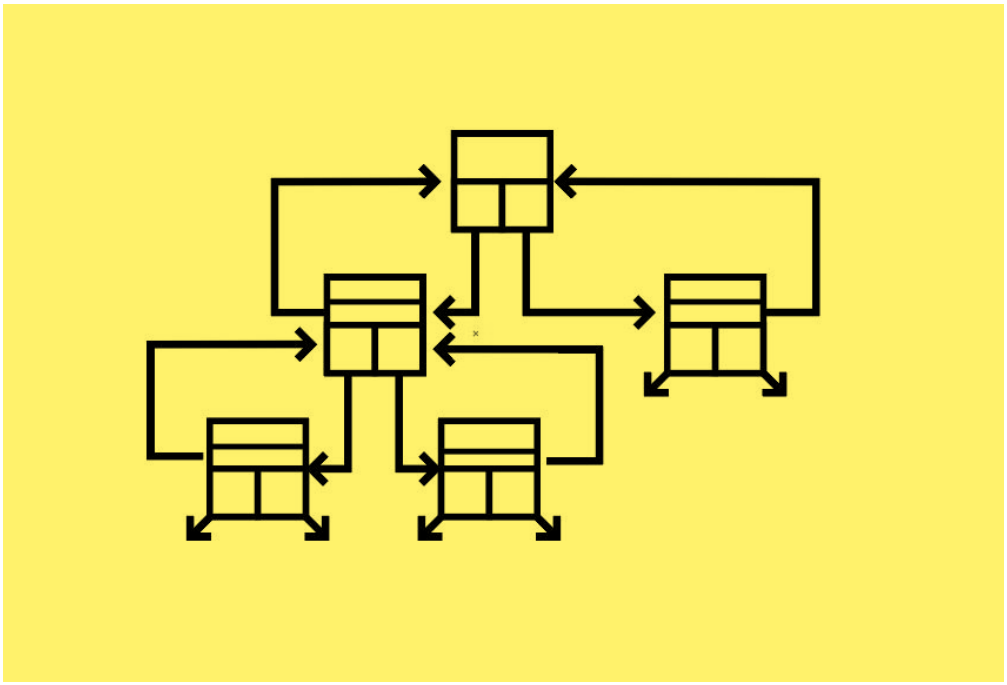
рис. 3

Описание основных функций и структур.

1. addAtom()

Объявление функции:

```
int addAtom(int atomNumber, int atomLeftNumber, int atomRightNumber, int atomArmLenght, int atomKey,  BinaryScales *&binaryScales)
```

Добавление элемента в общую иерархию.

2. balance()

Объявление функции:

```
bool balance(BinaryScales *&binaryScales)
```

Функция проверяет заданное коромысло на сбалансированность (выдает значение true, если коромысло сбалансировано, и false в противном случае).

3. backToZero()

Объявление функции:

```
auto backToZeroAtom(BinaryScales *&binaryScales)
```

Функция возвращает указатель на нулевой элемент.

4. printDepth()

Объявление функции:

```
int printDepth(BinaryScales *&binaryScales, int i)
```

Функция печатает список в соответствии с глубиной нахождения элементов, I – итератор глубины.

5. deleteAtoms()

Объявление функции:

```
int deleteAtoms(BinaryScales *&binaryScales)
```

Функция высвобождает память, выделенную для элементов бинарного коромысла.

6. BinaryScales

```
struct BinaryScales {
    BinaryScales(int atomNumber_, int atomLeftNumber_, int atomRightNumber_, int atomArmLength_, int atomKey_) :
                atomNumber(atomNumber_), atomLeftNumber(atomLeftNumber_), atomRightNumber(atomRightNumber_), atomArmLength(atomArmLength_), atomKey(atomKey_),
                isBalanced(true), isChecked(false), prevAtom(nullptr), leftAtom(nullptr), rightAtom(nullptr){};
        int atomNumber,
            atomRightNumber,
            atomLeftNumber,
            atomArmLength,
            atomKey;
        bool isBalanced, isChecked;
       BinaryScales *prevAtom, *leftAtom, *rightAtom;
};
```

Описание структуры подробнее см. в п. Описание алгоритма (стр. 3).

Тестирование программы.

Входные данные:	Выходные данные:
0 1 4 1 2 3 1 0 2 0 0 1 2 3 0 0 1 2 4 5 6 1 0 5 0 0 3 1 6 0 0 1 3	balanced 0 -1 --2 --3 -4 --5 --6
0 1 4 1 2 3 1 0	balanced 0

2 0 0 1 2 3 0 0 1 2 4 5 6 1 0 5 0 0 3 1 6 0 0 1 3	-1 --2 --3 -4 --5 --6
0 1 2 1 0 0 2 3 2 0 0 6 1	balanced 0 -1 -2
0 1 6 1 2 5 1 0 2 3 4 1 0 3 0 0 2 1 4 0 0 1 2 5 0 0 3 1 6 7 8 2 0 7 0 0 1 1 8 0 0 1 1	balanced 0 -1 --2 ---3 ---4 --5 -6 --7 --8
0 1 4 1 2 3 2 0 2 0 0 1 1 3 0 0 1 1 4 5 6 1 0 5 0 0 3 1 6 7 8 1 0 7 0 0 2 1 8 0 0 1 2	balanced 0 -1 --2 --3 -4 --5 --6 ---7

	---8
0 1 2 1 0 0 4 5 2 0 0 2 1	not balanced 0 -1 -2
1 1 2 1 0 0 1 2 0 0 2	error: wrong zero atom data
0 1 2 1 2 2 1 2 2 2 0 1 2	error: breaking ties
0 1 2 1 2 3 4 5 2 4 5 6 7 3 1 2 4 5	error: wrong hierarchical list!

Выводы.

Был создан специальный иерархический список, в котором использовались некоторые принципы двусвязных и односвязных списков. Было получено понимание POD – типов, конструкторов и условного стиля написания программ на языке программирования C++.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

```
#include <iostream>
#include <fstream>
#include <sstream>
#include <vector>

struct BinaryScales {
    BinaryScales(int atomNumber_, int atomLeftNumber_, int atomRightNumber_,
int atomArmLength_, int atomKey_) :
                 atomNumber(atomNumber_),      atomLeftNumber(atomLeftNumber_),
atomRightNumber(atomRightNumber_),               atomArmLength(atomArmLength_),
atomKey(atomKey_),
                 isBalanced(true),     isChecked(false),     prevAtom(nullptr),
leftAtom(nullptr), rightAtom(nullptr){};
        int atomNumber,
            atomRightNumber,
            atomLeftNumber,
            atomArmLength,
            atomKey;
        bool isBalanced, isChecked;
       BinaryScales *prevAtom, *leftAtom, *rightAtom;
};

bool balance(BinaryScales *&binaryScales) {
    if         (binaryScales->leftAtom->leftAtom         !=         nullptr
&& !binaryScales->leftAtom->isChecked) {
        binaryScales =  binaryScales->leftAtom;
        balance(binaryScales);
    }
    else     if     (binaryScales->rightAtom->rightAtom     !=     nullptr
&& !binaryScales->rightAtom->isChecked) {
        binaryScales = binaryScales->rightAtom;
        balance(binaryScales);
    }
    else{
        binaryScales->leftAtom->isChecked                                  =
binaryScales->rightAtom->isChecked = true;
        int    torqueLeft    =    binaryScales->leftAtom->atomArmLength    *
binaryScales->leftAtom->atomKey;
        int   torqueRight   =   binaryScales->rightAtom->atomArmLength   *
binaryScales->rightAtom->atomKey;
        binaryScales->atomKey    =    binaryScales->leftAtom->atomKey    +
binaryScales->rightAtom->atomKey;
        binaryScales->isBalanced   =   torqueLeft   ==   torqueRight   &&
binaryScales->rightAtom->isBalanced && binaryScales->leftAtom->isBalanced;
        binaryScales->isChecked = true;
        if         (binaryScales->atomNumber         ==         0         &&
binaryScales->leftAtom->isChecked  &&  binaryScales->rightAtom->isChecked)  return
binaryScales->isBalanced;
        binaryScales = binaryScales->prevAtom;
        balance(binaryScales);
    }
}

auto backToZeroAtom(BinaryScales *&binaryScales){
    if (binaryScales->atomNumber == 0) return binaryScales;
    else {
        binaryScales = binaryScales->prevAtom;
        backToZeroAtom(binaryScales);
    }
```

```
    }

    int printDepth(BinaryScales *&binaryScales, int i){
        if(binaryScales->leftAtom == nullptr && binaryScales->rightAtom ==
nullptr){
            for(int j=0;j<i;j++) std::cout <<"-";
            std::cout<<binaryScales->atomNumber<<std::endl;
            return 0;
        }
        for(int j=0;j<i;j++) std::cout <<"-";
        std::cout<<binaryScales->atomNumber<<std::endl;
        i++;
        printDepth(binaryScales->leftAtom, i);
        printDepth(binaryScales->rightAtom, i);
    }

    int deleteAtoms(BinaryScales *&binaryScales){
        if(binaryScales->leftAtom == nullptr && binaryScales->rightAtom ==
nullptr) {
            delete binaryScales;
            return 0;
        }
        deleteAtoms(binaryScales->leftAtom);
        deleteAtoms(binaryScales->rightAtom);
        delete binaryScales;
    }

    int addAtom(int atomNumber, int atomLeftNumber, int atomRightNumber, int
atomArmLenght, int atomKey,  BinaryScales *&binaryScales) {
        auto *tmpBinaryScales = new BinaryScales{atomNumber, atomLeftNumber,
atomRightNumber, atomArmLenght, atomKey};
        if (binaryScales->atomLeftNumber == tmpBinaryScales->atomNumber) {
            tmpBinaryScales->prevAtom = binaryScales;
            std::cout << "pointer left" << std::endl;
            std::cout << "pointer on atom # " << tmpBinaryScales->atomNumber <<
std::endl;
            std::cout << std::endl;
            binaryScales->leftAtom = tmpBinaryScales;
            binaryScales = binaryScales->leftAtom;
            return 0;
        }
        else if (binaryScales->atomRightNumber == tmpBinaryScales->atomNumber)
{
            tmpBinaryScales->prevAtom = binaryScales;
            std::cout << "pointer right" << std::endl;
            std::cout << "pointer on atom # " << tmpBinaryScales->atomNumber <<
std::endl;
            std::cout << std::endl;
            binaryScales->rightAtom = tmpBinaryScales;
            binaryScales = binaryScales->rightAtom;
            return 0;
        }
        else {
            binaryScales = binaryScales->prevAtom;
            if       (binaryScales->prevAtom       ==       nullptr       &&
(binaryScales->atomLeftNumber      !=      tmpBinaryScales->atomNumber      &&
binaryScales->atomRightNumber != tmpBinaryScales->atomNumber)){
                std::cout << "error: incorrect data order" << std::endl;
                return 1;
            }
            std::cout << "pointer back" << std::endl;
            std::cout << "pointer on atom # " << binaryScales->atomNumber <<
std::endl;
            std::cout << std::endl;
```

```
            addAtom(atomNumber, atomLeftNumber, atomRightNumber, atomArmLenght,
atomKey, binaryScales);
        }
        return 0;
    }

    bool checkAtomData(std::vector<int>& dataAtom){
        std::cout << dataAtom[0] << std::endl;
        bool check = true;
        if (dataAtom[0] != 0 && dataAtom.size() == 3) {
            std::cout << "error: wrong zero atom data" << std::endl;
            check = false;
        }
        else if (dataAtom.size()!= 3 && dataAtom.size()!=5){
            std::cout << "error: incomplete data" << std::endl;
            check = false;
        }
        else if ((dataAtom[1] == 0 && dataAtom[2] != 0) || (dataAtom[2] == 0 &&
dataAtom[1] != 0)) {
            std::cout << "error: breaking ties" << std::endl;
            check = false;
        }
        else if (dataAtom[1] == 0 && dataAtom[2] == 0 &&  dataAtom[4] <= 0 &&
dataAtom[0] != 0) {
            std::cout << "error: zero load weight" << std::endl;
            check = false;
        }
        else if (dataAtom[3] <= 0 && dataAtom[0] != 0) {
            std::cout << "error: zero arm length" << std::endl;
            check = false;
        }
        return check;
    }

    int main() {
        std::string testFileName, resultFileName, line, number;
        std::cout << " Enter test-file location: " << std::endl;
        std::cin >> testFileName;
        std::ifstream testFile;
        testFile.open(testFileName);
        if (!testFile.is_open()) {
            std::cout << "error: test file is not open" << std::endl;
            return 0;
        }
        std::cout << " Enter where to save results (location with <name>.txt):
" << std::endl;
        std::cin >> resultFileName;
        std::ofstream resultFile(resultFileName);
        if (!resultFile.is_open()) {
            std::cout << "error: result file is not open" << std::endl;
            return 0;
        }
        int lineCounter = 0;
        while (!testFile.eof()) {
            getline(testFile, line);
            lineCounter ++;
        }
        if ((lineCounter - 1) % 2 != 0) {
            std::cout << "error: wrong hierarchical list!" << std::endl;
            return 0;
        }
        testFile.close();
        testFile.open(testFileName);
        getline(testFile, line);
```

```
        std::istringstream issZero(line);
        std::vector<int> dataAtomZero;
        dataAtomZero.clear();
        while (issZero >> number)
            dataAtomZero.push_back(std::stoi(number));
        if (!checkAtomData(dataAtomZero)) return 0;
        auto *binaryScales = new BinaryScales{dataAtomZero[0], dataAtomZero[1],
dataAtomZero[2], 0, 0};
        while (!testFile.eof()) {
            getline(testFile, line);
            std::istringstream iss(line);
            std::vector<int> dataAtom;
            while (iss >> number)
                dataAtom.push_back(std::stoi(number));
            if (!checkAtomData(dataAtom)) return 0;
            else {
                addAtom(dataAtom[0],  dataAtom[1],  dataAtom[2],  dataAtom[3],
dataAtom[4], binaryScales);
             }
            resultFile << line << std::endl;
        }
        backToZeroAtom(binaryScales);
        if (balance(binaryScales)){
            std::cout << "balanced" << std::endl;
            resultFile << "balanced" << std::endl;
        }
        else{
            std::cout << "not balanced" << std::endl;
            resultFile << "not balanced" << std::endl;
        }
        std::cout << std::endl;
        std::cout << "depth of atoms" << std::endl;
        printDepth(binaryScales, 0);
        backToZeroAtom(binaryScales);
        deleteAtoms(binaryScales);
        resultFile.close();
        return 0;
    }
```